Mykhailo Danylov

Patient Engagement Platform for Hospitals

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Thesis

6 October 2017

Helsinki Metropolia University of Applied Sciences

Author(s) Title Number of Pages Date	Mykhailo Danylov Patient Engagement Platform for Hospitals 38 pages 6 October 2017
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Timo Salin, Senior Lecturer

The main goal of the final year project was to develop and deploy software product for Buddy Healthcare Oy. The solution is a web application dashboard which is used in hospitals by nurses or other personnel to monitor the process of preparation for the surgery or recovery after the procedure by patients.

During the project, there was a preliminary research done on modern technologies in order to make a rational strategic decision regarding the technology stack, especially for the front-end development. Also, the importance of software development process was taken into consideration and final product was thought through starting from technical requirements received from business side and finishing with automatic testing and deployment pipeline before the software is released to production where users can continue using it without any disturbances.

As a result, a web application dashboard was developed for Buddy Healthcare Oy. The product is a single page web application developed with Angular framework. The dashboard allows nurses to monitor most of the interactions of patients with the patient mobile app. The dashboard provides multiple features such as creating new procedures for patients, reviewing different types of questionnaires submitted by patients, checking if important information was read by patients, modifying instructions content, sending reminders to patients, rescheduling and cancelling of the procedures.

In conclusion, for the past year there were multiple pilot projects conducted with different hospitals in Finland and the feedback was positive. Moreover, there are a few hospitals which have already taken the dashboard into daily use and the development of the product continues according to their feedback.

Keywords	healthcare, hospital, surgery, patient engagement, angular, web application

Contents

1 Introduction

Nowadays scientific and technical progress is moving forward faster than ever. One of the main factors which has accommodated this speed and the rate of growth is the latest wave of the Industrial Revolution and its name is Digital Revolution. At first, it enabled scientists and engineers to gather and store data in a more efficient way for later analytics and calculations. However, it was not enough because different research groups were located significantly far away from each other to collaborate effectively. As a solution for this problem the first computer network was born in 1969. It was not yet the Internet as we know it today but by the end of that year four computers in different parts of the US were able to establish communication. [1]

Today, we can hardly imagine our lives without the Internet. We use it to search, to store and share information every day. There are more websites on the Internet than the total population on Earth. More specifically, Finland became the first country in the world in 2010 to make an internet connection a legal right for every citizen [2]. Consequently, it played an important role in boosting mobile data consumption and by the year 2016 the level reached, according to multiple resources over, 7 GB on average per SIM card [3].

Another field in Finland where digitalization of information has left a significant mark is a health care sector. For example, a centralized service with public health records has already been in use for a few years. However, there is still room for improvement and business opportunities in health care. Buddy Healthcare is a startup company which tries to address a few important problems in hospitals' everyday activities. First, the company offers a mobile application solution to guide patients through the surgery preparation process as well as to provide postoperative instructions to help patients to recover more easily. Second, Buddy Healthcare provides a web dashboard solution for hospitals to monitor patients' performance and help to concentrate on and bring attention to the patients who need to be cared for the most. The latter solution is the main topic of this thesis.

There are a few problems which Buddy Healthcare tries to solve by developing their solution. The first one is communication between the hospital and the patient as well as within the hospital itself. The second problem is pre-surgery adherence. According to

Buddy Healthcare up to 17% of operations are getting cancelled in the last moment before the operation. Thus, the web dashboard will allow to minimize this risk by monitoring the patients' preparation process more carefully. The third one is postoperative care. While hospitals lack the resources for monitoring patients' recovery process, the data gathered in the form of a dashboard can help to follow up and identify patients who need attention the most. The last one is the overall quality of care. Incorrect clinical data is a big risk for patient health. Therefore, informed pre-surgery preparation is highly important for patient safety and well-being.

The goal of this thesis is to build a patient engagement platform for hospitals for monitoring patient preparation process for the surgery as well as post-surgery recovery.

2 Theoretical background

Nowadays, the field of web development has grown in size dramatically. However, one tool has remained the same over more than 20 years and it is the programming language of the web - JavaScript. Until now, it is still the only programming language which runs on web browsers. For historical reasons JavaScript is a unique programming language since it is asynchronous by default. It means that software developers who use it have to acquire a different approach to write programs compared to developers who use languages with sequential execution such as C++, Java or Python. Moreover, decentralization of the Internet and openness of the web development community had led to the development of a vast variety of open source frameworks and tools which help to increase productivity and standardize software development practices for the web.

This chapter covers generic research done in the area of web software development in order to choose appropriate technologies to implement a dashboard for hospitals to monitor patients' surgery preparation and recovery procedures.

2.1 Client-side web frameworks

As of the time when this thesis is being written in July 2017, there are 4 major web-client side pieces of software available for use (the word "frameworks" is omitted here on purpose as the size of software packages varies dramatically and some are more likely to be called "libraries"). Their names are React, Vue, Angular and Meteor respectively in ascending order by the size of the software package.

This section provides a brief description and comparison of the abovementioned software packages. Furthermore, section 3.1 provides more explanation and reasons for choosing the Angular framework for developing the dashboard.

2.1.1 React

According to the React project home web page, it is a library for building user interfaces. Backed and developed by Facebook, this library has become the most

popular JavaScript library according to the number of stars it has on Github. One of the most common techniques in front-end development which React leverages is a component-based approach. It is a simple yet powerful paradigm because it is focused on building small reusable blocks which are independent of each other but when put together could produce a user interface of any arbitrary complexity. However, there are a few features which are unique to React library. [4]

The first one is JSX, a JavaScript extension syntax used in React. It may look similar to HTML or some templating language but it is neither of those because it is built on top of JavaScript. For example, a simple heading element can be declared as illustrated in the listing 1. [4]

```
const greeting = <h2>Hello, world!</h2>;
```

Listing 1. React JSX syntax example

The second interesting concept that is used in React is elements immutability. Once a component is rendered, it cannot be changed anymore. On the other hand, if we need to update an element it should be rendered again by React. It might seem as an overhead to performance but the library actually compares a new element with its previous version and updates only those parts of the element DOM (Document Object Model) tree which have been modified. [4]

Last but not the least, currently React has the largest community of front-end developers in the world. However, the library itself offers only a limited set of tools for a developer and it is easily extendable by other libraries. Thus, in order to build a product with React, a software development team has to carefully choose and decide on the tool set they are going to use.

2.1.2 Vue

Vue to the contrary of React, is positioned as a progressive framework according to the vuejs.org website. On one hand, a more closer look at the documentation shows that Vue covers similar concepts of component-based development and renders elements with the help of virtual DOM. On the other hand, there is one interesting part in Vue implementation which differs more from a developer perspective. It is the way

framework defines and handles lifecycle hooks of a component. Vue has a more extensive set of lifecycle hooks which includes 9 states such as beforeCreate, created, beforeMount, mounted, beforeUpdate, updated, beforeDestroy and destroyed, thus allowing for more flexibility in defining the behavior of building blocks. [5]

Another unique feature of Vue framework is the concept of Mixin. It is a way to define a common set of properties and functionality which then can be extended by any component. In general, mixins have been imbedded already in some programming languages such as Python, Ruby or Scala. However, the concept is relatively new considering web front-end frameworks. [5]

In addition, Vue has a large community of developers around the world who contribute to both to the framework itself as well as by developing additional extensions and libraries which can enhance the original framework.

2.1.3 Ember

Ember is one of the biggest front-end frameworks. Therefore, it covers every aspect of the development process and provides a wide range of tools to accomplish the most of required tasks. Some items from the toolset of Ember are Ember CLI (Command Line Interface), Routing, Template Engine, Data layer and Ember Inspector. Ember CLI is a toolkit to speed up development and ease the build process of a web application. Routing is one of the main building blocks of any web application and it enables to derive an application state from a URL (Uniform Resource Locator). the Template engine which Ember uses is named Handlebars and it makes it easy to update data in real-time in HTML (HyperText Markup Language). The Data layer in Ember provides a consistent way of communicating with the server and managing the state of the application. Ember Inspector is an additional tool which helps to debug Ember applications directly in the browser. [6]

Figure 1 describes how the ember framework workflow is organized in detail. First, user inputs a URL in the browser which triggers a specific route to be loaded. Then, the Route Handler is called which can in turn extract needed data from Model or trigger an API (Application Programmable Interface) to a server to fetch required data and save it locally to web storage. After that data is passed to a template where any component

can make use of it.

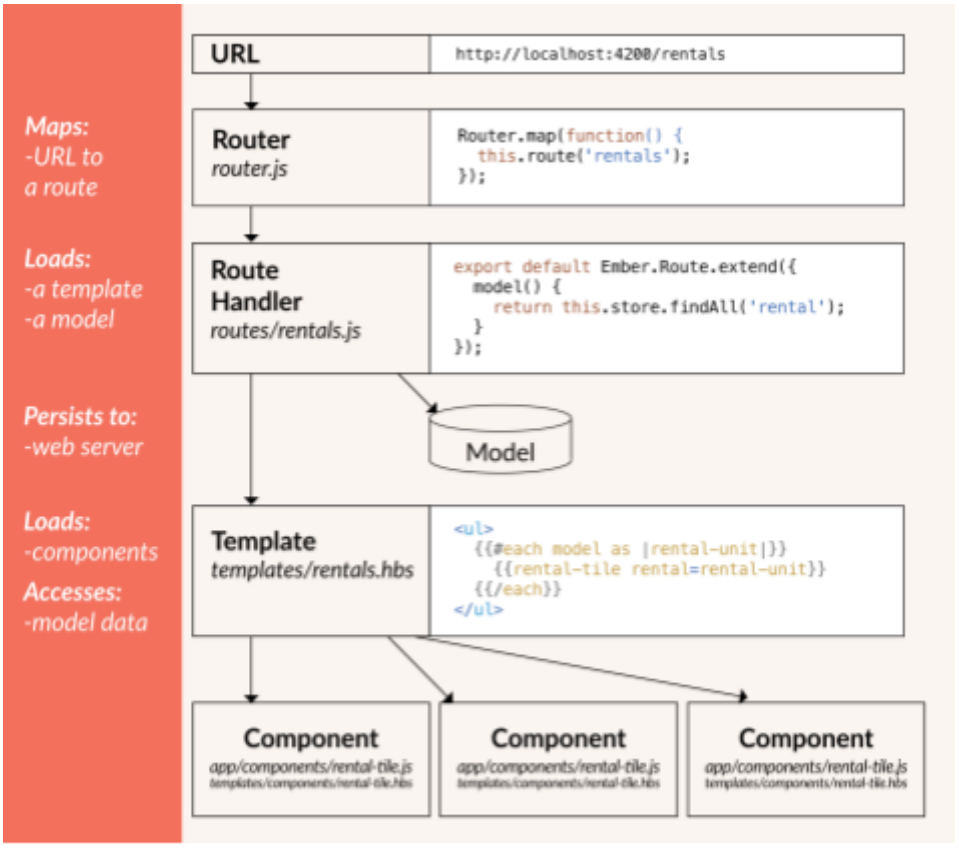

Figure 1. Ember Framework Architecture. Copied from Ember.js Documentation [6]

Overall, Ember is a strongly opinionated framework which has a rather steep learning curve considering all different tools included in the package from the beginning. On the other hand, once a developer feels comfortable with Ember, it will be easy for him/her to switch between different projects which use the same framework.

2.1.4 Angular

The first version of the Angular framework, namely Angularjs, was written and released in 2010. At that time, it was one of the first attempts to consolidate web front-end development and give software developers a set of tools, which could be used in a variety of projects. Moreover, it was an attempt to develop the first framework which did not require any additional libraries to build a full-featured interactive web application while staying flexible enough to be easily extendable.

Later on web technologies evolved and front-end software development moved towards a component-based approach. At that time the Angular team from Google announced a complete rewrite of the framework and released Angular 2 in September 2016, known today as just Angular. Similar to a previous version it offers a complete toolset to develop a full-featured web application without additional libraries used. However, it takes almost no effort from a developer to extend Angular by adding any additional software package.

Similar to other modern JavaScript libraries and frameworks, Angular is focused on a component-based approach. However, there are a few significant differences as compared to React or Ember. An overall architecture of the frameworks is illustrated in figure 2.

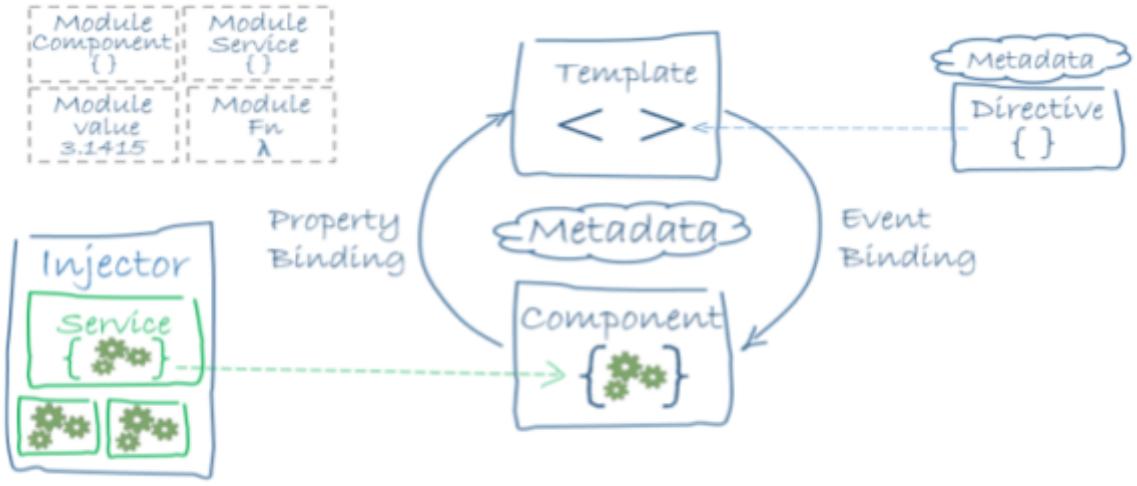

Figure 2. Angular overall architecture. Copied from Angular Documentation [7]

Components, Directives as well as Services are the main building blocks in Angular and they are grouped into modules. Modules in turn can be imported by other Modules. As for the Component itself, it consists of Typescript class with properties and methods and it is bind with HTML template and CSS (Cascading Style Sheets) styles over class metadata. The binding allows to use component class public properties and methods directly in templates. There are a few key advantages of adopting this framework for web-based projects from medium to large size.

First, the latest version of Angular was written using a superset of JavaScript - Typescript. Latter, originally developed by Microsoft as an open-source project, has attracted a large number of contributors around the world. As the name implies, Typescript brings type checking to JavaScript. The language is compiled and it means that variable types, return values from functions class properties and methods will be checked during compilation even before the code is executed. Moreover, Viktor Savkin pointed out that the biggest advantage of Typescript is tooling. It offers advanced refactoring, autocompletion and navigation which are in turn essential for any large project. [8.]

The first tool that a developer gets to use with Angular is its CLI. This package allows to scaffold the project structure as well as different parts of the application such as components, directives, pipes, services and modules. CLI will also create a file with a skeleton for a test suite for each of the generated files. Moreover, this package helps the development process by running a live server and watching for application file changes and restarts the server whenever any code has been changed. In addition, it provides a powerful integrated building and bundling of the whole application with multiple stages of optimizations.

Second, Angular Router offers a powerful and highly configurable module to set up a navigation for a web application. However, the most significant feature that it provides is routes lazy loading. The main concept of it lays in dividing the application into feature modules and let the router load them from the server only when they are actually used by the end user. Therefore, at the first page load only one main module is going to be requested from the server and all other modules with all their component dependencies will be loaded from the server as soon as the user starts interacting with that feature module. This feature would be beneficial for any type of application to speed up the startup time as well as increase security of the application.

Overall, Angular provides a rich framework and ecosystem for developing enterprise level web applications. The framework helps to standardize the development process by using Typescript as well as offers a high level of automation tool for testing and deployment and provides a high level of security.

2.1.5 Meteor

As of the time of writing this paper, Meteor is the only full-stack JavaScript platform for building web applications using only JavaScript. It means that a single language can be used to develop server software as well as a front-end client. Moreover, even mobile applications can be developed with little effort with Meteor. However, there are a few main features of the platform which are worth mentioning separately.

First is the way Meteor handles data transmission of the application. By default, any app build with the framework is fully reactive. That implies that any client 'reacts' to any data change on the server and updates user interface accordingly. [9]

Second is the ability to develop both back-end and front-end in a single language - JavaScript. Moreover, some part of the codebase can be even reused between them. It presents a great opportunity within a development team because it becomes easier for software engineers to switch roles and the same people can accomplish tasks for both sides. [9]

However, the most interesting feature of Meteor is the way data is stored and updated on the browser. As defined in documentation, the server leverages MongoDB for persisting data and the front-end has a local shortened copy of that database which gets synchronized with the main database on the backend. Thus, it implies that on the client side we have a full-featured browser database which replicates MongoDB APIs. [9]

Overall, Meteor is a solid and full featured platform for developing full-stack JavaScript applications. However, it might not be suitable if a company has chosen another development stack for the server side. Also, it has other restrictions regarding storing data because currently only MongoDB is a supported database.

2.2 Data storage patterns

Data has become a valuable asset in recent years. The more information a company has about its users and the ways its products are used, the better analysis it can make

for improving its situation on the market. However, there are many ways to store data in web applications and this chapter describes the most common ones.

2.2.1 Web storage

Web Storage API is a part of EcmaScript specification and it is implemented by the most popular browsers such as Chrome, Firefox, Safari, Edge and Internet Explorer. It is an intuitive and easy to use in-browser storage which allows to manipulate data for a certain domain. Nowaday, there are two types of Web Storage available on the browser. They are window.localStorage and window.sessionStorage respectively [10].

First, the localStorage property of the window object allows developers to access a plain JavaScript object. The latter does not have any expiration date and all the data will be stored in the object permanently until it is removed manually using browser settings. localStorage is an implementation of Storage interface and has a simple API for adding or removing key value pairs as we can see from the listing 2. [10]

```
localStorage.setItem('isThesisDone', true);
let thesisStatus = localStorage.getItem('isThesisDone');
```

Listing 2. Web Storage API example

Second, sessionStorage property is a similar implementation of Storage interface. The only different part is that all data for a particular domain is removed as soon as browser window is closed. In addition, it is important to note that data handled by both storages is specific to the protocol of the page. [10]

However, there are a couple of concerns regarding Web Storage. First, there are far too many browsers and not all of them might have implemented the APIs or they might have different implementations and developers have to take care of compatibility issues. Another issue concerns “Incognito” or “Private Browsing” mode, which treats the storage differently depending on the browser. However in most cases all the data is wiped when user closes the browser window.

Overall, Web Storage API provides an easy and convenient way of storing key value pairs in the browser memory. However, it might cause data loss in special cases when

user opens browser in “Incognito” mode. Also the storage available can be limited by browsers and the security of the data is dependent on each browser implementation and therefore not under the control of the application developer.

2.2.2 Flux

One of the most recent trends in managing a data flow in web applications is the Flux architecture. It was initially developed by Facebook and later on widely adopted by the front-end community throughout the world. Flux is a redesign of the MVC (Model View Controller) pattern with one main difference and it is a unidirectional data flow which Flux utilizes. In figure 3 we can see data transmission paths according to the Flux pattern.

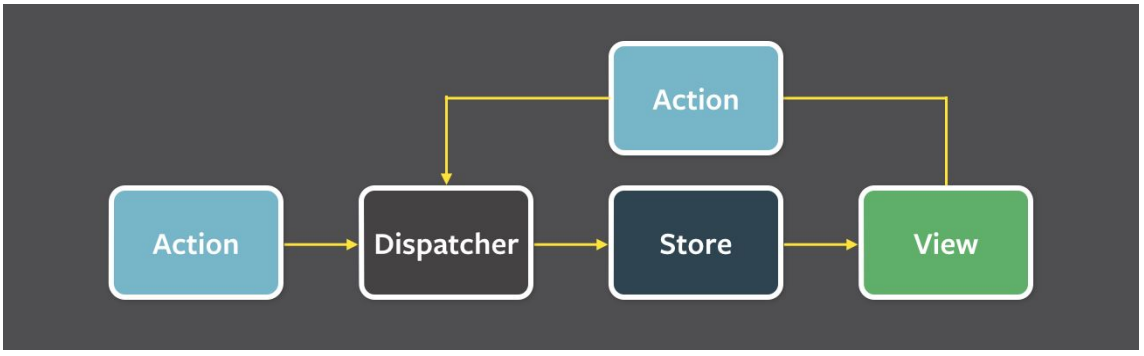

Figure 3. Flux unidirectional data flow. Copied from Flux Documentation [11]

There are 4 main building blocks in the architecture: Action, Dispatcher, Store and Views. Any interaction starts with a View when the user of our application performs any action. User interaction with the UI triggers an Action which then reaches a centralized Dispatcher. Latter listens for an Action to be triggered and updates data in Store. It is important to mention that only the Dispatcher can perform or delegate to other application logic any data modifications. Then a View subscribes for any changes of some particular piece of data in Store and always gets the latest version of data once it is available. Also, Actions can be triggered by other logic of the application, not necessarily by a View. [11]

Overall, Flux offers a new way of thinking about data storage and flow for web applications. It helps to manage the application state in a more predictive manner, which makes it easier to reason about the app logic. Also, the pattern makes it easier to

add new features as an application grows in size or onboard a new developer because the data flow is unidirectional and straightforward to follow.

2.2.3 Redux

Redux is probably the most popular JavaScript library, a successor of Flux, for managing data in web applications nowadays. The library follows three main principles. First, an application state is stored in a single tree state represented by plain JavaScript object and called Store. This state becomes a single source of all the data in the application which makes it easier to track back the changes in debugging process. Second, the only way to change a state is to emit an action which is a description of what needs to be updated in Store. In this way it is not possible to have any race conditions when all actions are handled in a centralized manner. In addition, the pattern ensures that views or network responses cannot modify Store state directly. Third, modifications to Store are done with the help of pure functions called Reducers. It means that a Reducer does not modify the existing state but instead returns a copy of the modified state. [12]

Redux itself is small however it encourages the use of extensions and plugins. For example, the library does not dictate which library to use to store the data. Therefore, any library is suitable as long as it supports immutability of data because it is not allowed to mutate the state of Store in Redux. [12]

Overall, Redux is a logical evolution of Flux architecture. First, it simplifies API of the library and decouples the data updates from Stores in Flux to Reducers in Redux. Also, the latter does not have the concept of Dispatcher because it relies on pure reducer functions instead of event emitters.

2.2.4 In-browser databases

Evolution of web technologies could not leave out solid and mature approaches for data storage developed in the past such as SQL (Structured Query Language) databases. Consequently, new ways of storing data in a browser emerge which were not available previously for the web while browsers become more performant and

capable. As a result of such advancements, a new category of data storage technologies was born - browser database. As of time of writing this thesis, a browser database is defined as IndexedDB API standard by W3C (World Wide Web Consortium). It is important to mention that IndexedDB API is a part of Web API standard recommendation and supported by most of the modern browsers. Standard is defined as a low-level API for client-side storage for large sets of structured data or files. Similar to regular SQL (Structured Query Language) databases their browser versions support indexing for high-performance search as well as transactions. However, it is not a fixed column table but a key value storage which allows to retrieve data by a key. Also, all database operations are performed asynchronously in order not to block the main thread of the application and its UI. Over recent years there have been multiple attempts to build an abstraction layer on top of IndexedDB API and provide JavaScript API for web applications. Some of these libraries have gained popularity and approval by the web developers' community. [13]

Probably one of the most popular JavaScript database libraries is PouchDB. It helps developers to create web applications which work offline by utilizing browser storage. Moreover, it provides an easy way to synchronize data between a browser and a server whenever a user comes back online. PouchDB takes advantage of IndexedDB API but even though the library supports wide range of modern browsers it falls back to WebSQL if IndexedDB is not supported by the browser. Also, the library is platform-agnostic and works with any JavaScript framework or library. [14]

Another powerful in-browser database is called Lovefield and it is developed by Google. The library implements relational database in pure JavaScript and provides syntax similar to SQL. Lovefield supports most desktop browsers. The main idea behind the library is to define a database schema at first. Then, the schema cannot be altered anymore and all operations are done on the database instance. [15]

Overall, one of the key benefits of using an in-browser database is the ability to process fast large amounts of structured data. Many real-time applications would benefit from this approach of handling data sets on the client side. However, storing personal user information in a potentially insecure environment is not ideal for healthcare-related applications unless information in the database is encrypted.

2.3 Data transmission techniques

Network communication is an essential part of any web application. As a matter of fact, web applications would not exist if there was no Internet because they were a consequence of the development of the Internet. In the early days of web applications the only way to display a web page was to request that page from a server. Then, the content was rendered on the server side into HTML and delivered to the client. Since then, the technologies around data transmission have evolved dramatically. Thus, the most common ways to handle data transmission are discussed in this section.

2.3.1 Server-side rendering

Nowadays, server-side rendering is still widely used in web applications. However, as web technologies has evolved the ways of handling this process has changed as well. Therefore it is unlikely to find any HTML file on a server which is sent as it is without any modifications to the client side. Many templating languages has emerged which try to improve the way HTML is written and organized on the backend. One of the most popular templating engines are Mustache and Handlebars and they both have a similar syntax illustrated in the listing 3.

```
<div class="wrapper-class">
    <h1>{{headerVariable}}</h1>
    <div class="body-class">
        {{bodyVariable}}
    </div>
</div>
```

Listing 3. Handlebar template syntax example

As we can see from listing 3, the template language looks similar to regular HTML except for double figure brackets containing variable names. The content inside these brackets is rendered in the final HTML which is presented to end-user through the web browser. Other useful features of Handlebars include block expressions which allow to loop through lists of data and partials, helping to reuse parts of HTML template as a part of other templates. [16]

Overall, server-side rendering has evolved and with the help of template engines allows developers to write less code but in more efficient ways. However, this approach might not be enough to build highly interactive web applications which require JavaScript to run on the browser.

2.3.2 REST

REST (Representational State Transfer) is a set of rules or architectural design for web client-server communication. REST was originally defined in a PhD thesis written by Roy Fielding in 2000. However, it took a few years for software development industry to evaluate and adopt concepts of RESTful applications. Nowadays, it is probably the most widely used architecture in web development. There is a set of milestones or architectural constraints which define REST. [17]

First constraint, server and client are treated as separate entities and therefore can be developed in parallel. In addition, this separation of concerns helps to decouple backend logic from front-end logic. Second constraint is interaction related and it considers server to be stateless. Thus, all context information should be stored on the client side and each request to the server should contain all information needed for a request to be interpreted correctly by the server. Third, the server should define whether the response can be cached on client side and the latter should be able to store data locally and make a request for new data only when the cache has expired. Fourth constraint explains that the client cannot know exactly if it makes a request to the final destination server or to any intermediate server therefore introducing the concept of layered system. Fifth, a client application can be extended by requesting additional parts from server. Last is the uniform interface milestone and it is an essential part of the design for any RESTful interface. The latter implies that the interface is decoupled from its internal server implementation and does not necessarily correspond to how data is structured in the database and might represent data aggregated from multiple services, therefore improving the evolvability of the system as a whole. [18]

Overall, REST is the number one option when designing a web application. Over the years it has proven to be a simple yet reliable way of implementing client-server

communication.

2.3.3 Web Socket

A relatively new way of client-server communication is The WebSocket Protocol standard. It was proposed in 2011 by IETF (Internet Engineering Task Force) [19]. Since that time most of the modern browsers have implemented WebSocket API. It is an advanced technology which offers a long-lasting interactive communication session between a browser and a server. The main advantage over RESTful communication is that once communication is established, the client does not necessarily have to poll for any data because communication is handled in both directions. Thus, the client can send data to the server as well as server can push data to the client without a request. The main interface in the browser is WebSocket and it provides functionality for creating and managing a connection as well as receiving and sending data. One simple example of how the communication can be established using WebSocket is shown in listing 4.

```
// create websocket instance
let socket = new WebSocket('ws://api.buddyhc.com/websocket');

// sending data to server after connection is established
socket.onopen = (event) => {
   socket.send(JSON.stringify({message: 'Hello, World!'}));
}

// receiving data from server
socket.onmessage = (event) => {
   console.log(event.data);
}
```

Listing 4. WebSocket API example. Copied from MDN web docs [20]

At first, the client sends an HTTP request to server with Upgrade header specifying websocket. Therefore, server is notified about the client intent to switch to WebSocket protocol. Then, communication is switched If the protocol is supported by the server.

Thus, full-duplex communication is established between client and server. [20]

Overall, WebSocket can be considered a reliable modern technology. It enables real-time client-server communication and can improve user experience and significantly reduce network traffic and server load for a web application.

2.3.4 GraphQL

Probably the most recent advancement in client-server communication is a technology named GraphQL. It was first introduced by Facebook in 2015 and currently it is in a state of Draft RFC Specification. GraphQL takes a different approach at enhancing data communication for web applications. It is a query language which is used both on client and server side. On the client, the language is used to construct queries. On the server, a runtime interprets those queries and returns a result which is similar to that of a query, typically in JSON(JavaScript Object Notation) format. [21]

First, defining a GraphQL API starts with creating a service. The library a type system and therefore a schema with types and fields should be defined on server side. Then, function should be created for each field on a type defining how the value of the field can be obtained. Once the service is running on a server, it can receive queries. Each query is validated against its schema before it is evaluated and JSON output is produced. [22]

Second, GraphQL queries can be constructed on the client side and send to the server. The main advantage of using this query language is that response object structure is the same as the query structure itself. Therefore, it is conceptually more simple for a developer to construct a query. An example of such a query and its result returned by the server is shown in listing 5. [22]

```
# GraphQL query
{
  user {
    name
    patients {
      name
```

```
    }
  }
}
# GraphQL response
"data": {
  "user": {
    "mikedanylov",
    patients: [
      { "name": "patient1" },
      { "name": "patient2" },
    ]
  }
}
```

Listing 5. GraphQL query and response example. Modified from GraphQL Documentation [22]

Both the request query and JSON response have a similar structure and therefore could be more intuitive for a software developer to reason about.

Overall, GraphQL is a promising technology which can bring more flexibility for managing different backend technologies and then unifying them into a centralized GraphQL service interface. Also, it could increase the speed of the development process by providing an intuitive GraphQL query language.

3 Methods and materials

In the previous chapter a brief analysis was made of different technologies available nowadays for web development. A few user interface libraries and frameworks were discussed for building a general structure of the application. Then, various ways of storing and manipulating data within the application were described. At last, the most common patterns and techniques for data transmission and client-server communication were analysed.

This chapter is going to go deeper into the details of the technologies which were chosen to accomplish the main goal of this thesis - building a web application dashboard for hospitals to monitor patients and provide them with help and guidance in preparation and recovering after a surgery.

3.1 Selected framework

After careful consideration the Angular framework was chosen as a core for developing the target application. First, a significant benefit for choosing Angular is the fact that it uses Typescript programming language by default. In fact the framework itself was written in that language. Typescript is a superset of JavaScript. It has a compiler, it encourages developers to use OOP (Object Oriented Programming) style and its pattern, still allowing the programmer to apply functional paradigms where necessary. The language helps to keep the project code base well structured and easy to follow, therefore minimizing the time a new developer needs to join the team.

Second, the Angular framework has a built-in support for RxJS library which is a powerful tool for handling asynchronous code. Considering the fact that JavaScript is asynchronous in nature, being able to handle such operations is vital for the project. Moreover, even the Angular HTML templating engine has support for subscribing and unsubscribing for asynchronous values.

Third, a highly configurable Router module is included in the framework. The Angular Router has multiple features which were used in the application. For example, routes lazy loading allows to request separate modules from a server only when they are

being interacted with by a user. Therefore, allowing for faster first page loading time and saving bandwidth by not loading unused parts of the application. Another feature is route guards. They help to protect some parts of the interface which are not supposed to be used by a particular user or a group of users.

Fourth, the framework includes http module by default. The latter allows to make a highly configurable http request to a server by modifying headers and including different types of parameters and data. Also, the latest version of the module includes interceptors support which allows to insert a middleware logic in the http request pipeline. In addition, the Angular http emit events on request progress which can be useful in determining how long does it takes to get a response back.

Fifth, the Angular templating engine and data binding covers all the spectrum of the needed tools to present to the end-user a final interactive interface. Displaying data is made easy by component property one- or two-way data binding. Multiple structural mechanisms are available for improved code reuse such as *ngIf for condition comparison and *ngFor for looping over collections of data. In addition, all asynchronous subscriptions are handled by the templating engine which means observer objects do not have to be unsubscribed explicitly.

Overall, the Angular framework proved to be a predictable and reliable tool for developing the hospital dashboard.

3.2 Application architecture

First, In order to be able to formalize the application architecture the main data resource should be defined and the relationships between them. Later, after all needed data is known we can start dividing the application into modules by grouping related items together. Below we can see a list of the main data elements of the app.

1. User
2. Nurse
3. Patient
4. Hospital
5. Operation Template

6. Operation
7. Timeline Event
8. Cancellation Option
9. Cancellation Reason
10. Quiz
11. Quiz Item
12. Quiz Comment
13. Conversation Thread
14. Conversation Thread Message

In the coming subsections separate modules will be defined based on listed data models above.

3.2.1 Login module

From the list above, User is a person who logs in to the web application. Furthermore, both Nurse and Patient could be users. However Patient should not have access to the hospital dashboard. Therefore, these models can become a part of login module. The main purpose of the module is to give access to only authorized users as well as provide some additional security features such as 2FA (Two Factor Authentication). After the dashboard is loaded, the user is faced with login page which serves as an entry point to the application. At this stage no other modules have been loaded yet as a result of performance and security enhancements. A figure 4 illustrates the logic for the login process.

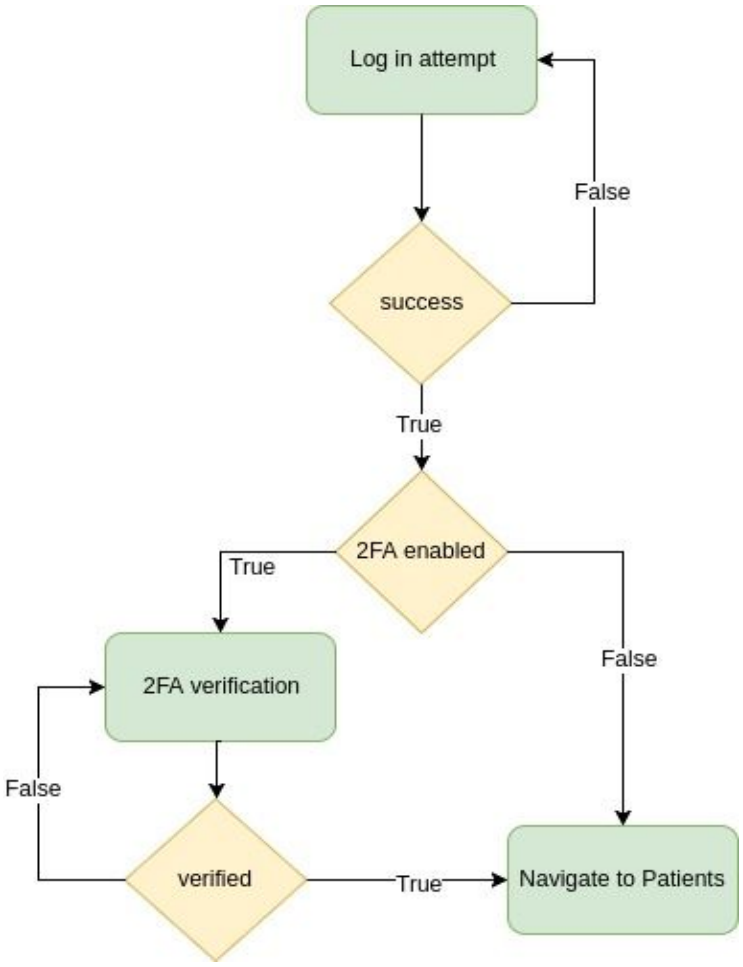

Figure 4. Dashboard login flow diagram

As the first stage of the flow diagram from figure 4, the user have to provide email and password and try to login. The second stage is needed only if user has 2FA enabled. Then he/she will have to provide a 2FA token and send verification request. After that, upon successful login user will be redirected to Operations page and the operations module will be requested containing all needed application logic to build the UI for Patients section.

3.2.2 Operations module

The Patients section is the most data intensive part of the application. Moreover, it is the main view which is shown to a nurse after the login process. Most of the important information about any operation is gathered in this view and displayed in a table format. There are a few models involved in building a data layer for the operations module. They are Operation, Patient, Timeline Event, Cancellation Option and Cancellation

Reason. Each of these models have a part of the UI dedicated to them. The main operations view is divided into two subpages - active operations and archived operations. Active operations page has three tabs for different types of filtering. Here a user can select to show all operations, pre-operations, or post-operations. Pre and post refers to the operation phase whether surgery is upcoming or has already been done. Intuitively, all operations tab combines both views for broader overview. Each tab has its own purpose and data in operations table is reflecting the change when the tab has been changed. However, there are common fields in the table as well, for example patient name, contact information, surgery date and type are common for most of the tabs.

The first important feature of pre-surgery section is notification alarm which indicates that patient has deviated from the instructions provided by the timeline of events and a nurse should take an action. The next step for a responsible hospital worker would be to check what exactly the alert is about. That information can be found from a timeline considering the fact that most of the application logic is based on performance of a patient in following up instructions provided in the timeline. The latter, is a collection of timeline events which have different types of content and require some interaction from a patient. Each row in a operations table can be clicked and another operation expanded view is presented to the nurse where he/she can choose between displaying only important timeline events or show entire timeline. Thus, by checking the timeline the user can recognize which timeline event has triggered the alert notification and act accordingly.

There are a few possibilities what can be done by nurse after the source of alert has been found. Timeline events are marked as acknowledged by patient when the task is accomplished. Furthermore, each event has 3 main dates. First, when it becomes active and available for acknowledgment. Second is a soft deadline for the task. Mobile application will indicate that the event is urgent and user have to proceed immediately. Also, push notifications are sent to remind patients of their to-do lists. Last comes an expiry date after which user is no longer able to complete the task in the application. The following figure 5 depicts the lifespan of a timeline event.

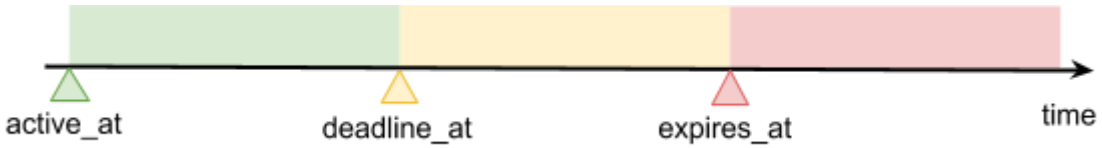

Figure 5. Timeline event lifecycle

From the figure 5 we can examine the time axis with 3 dates active_at, deadline_at and expires_at corresponding to timeline event activation date, deadline date and expiry date respectively. Also, the yellow and red areas indicate when a nurse might need to act upon certain event. It is possible for a nurse to send a manual push notification reminder in case when deadline is missed (while current date and time is within yellow area). However, later it becomes possible to acknowledge an event from a nurse dashboard when the expiry time has passed for an event (red area in figure 5). In the last case the nurse would need to contact the patient to confirm that the patient is still following instructions and is getting prepared for a surgery.

Another pair of features are operation rescheduling and cancellation. There could be a few use cases for operation rescheduling as both hospital and patient might have their reasons for changing date and time of an operation. Therefore, each operation expanded view has a toolbar at the bottom with a set of operation actions. One button is dedicated for operation rescheduling. When a nurse wants to change the date and time of the operation that button should be clicked on and a new modal window will pop up with simple instructions on how to proceed. The first step of the procedure is to select a new date and time. Then, the nurse must select reasons for operation rescheduling from a list of checkboxes as well as elaborate on the reasons selected. After that, confirmation for the action is sent to the server which in turn processes the request returns a new operation object back to the dashboard and sends push notification to mobile application to inform patient about date change. Furthermore, an operation can be cancelled in a similar way. A user needs to click on respective operation action and another modal popup for operation cancellation is shown. This time there is only one step to provide reasons for operation cancellation and to confirm the action. As for the mobile app users, they will receive a push notification about their operation being cancelled with reasons. However, the operation which has been cancelled does not belong to the active operations view and therefore is moved to archived cancelled operations tab. The latter tab contains a list of cancelled operations and when clicked on shows all the reasons for cancellations provided by the nurse.

3.2.3 New surgery module

Every operation creation process starts with New Surgery Module. It has a single page with three steps to complete new surgery creation. First, the nurse has to provide required information about new surgery: patient names, surgery type, date and time. Next follows a confirmation of the operation creation. Finally, the dashboard displays an activation code after new operation has been created on the backend. The code is supposed to be given to the patient who will use it in a mobile application to access the operation. After the operation is created it has a special status Internal and shown in the main operations view with status Activation Code. Also, a surgery can be created for both existing and new patients.

3.2.4 Forms module

Questionnaires play an important role in preparing patients for a surgery. There are a few types of forms which patients or their caretakers should fill out before or after the operation. As for the patient mobile application, forms are presented as a timeline events as well as a separate menu entry in the navigation bar so that users have an easy access to forms. Each form contains multiple questions of different types such as simple yes or no, single choice questions, multiple choice questions, free text questions as well as ratings and scales. Moreover, some questions might have an expected correct answer, which helps to evaluate a patient's readiness for upcoming procedure. As for the hospital dashboard, form menu item will load a forms module. The top level component of the module combines top toolbar with additional navigation and a list of questionnaires. Additional navigation between different types of forms as well as switch between active forms and archive is placed in the top toolbar. Then, below the toolbar two lists with filled and missed forms could be found. The difference between the lists is that acknowledged form should be checked and marked as reviewed by nurse. Also, there are alerts to notify of possible errors in filled forms. Regarding missed questionnaires, a nurse has the possibility to send manual push notification reminders to patients as well as accept a form without patient acknowledgement. Another view in archived forms has a single list with all forms which were filled in the past for already completed operations.

3.2.5 Conversations module

Conversation module is a lightweight implementation of a chat application. It enables a quick way of communication between a patient and hospital personnel. The main view of the chat consists of a list of current ongoing conversations with patients or their caretakers and a top toolbar with instant search for chat threads and the possibility to start a new thread with a patient. Moreover, a separate chat window is opened when a single chat thread is clicked. Currently, the messaging application supports sending text messages and images. Thus, text messages could provide a fast and easy way of clarifying some important details about the surgery or instructions in general and images could be needed by hospitals to provide proper guidance after the surgery.

3.2.6 Operations calendar module

The Calendar module is another way of tracking upcoming operations for hospital personnel. This view enables an easy way of keeping track of operations schedule. Moreover, there is a possibility to reschedule or cancel a surgery if needed. The main view has three tabs for more specific filters of operations by daily, by week or by month. Each of subviews can be navigated forward and backward. Also user can always return to the initial view which includes current date.

3.2.7 Settings module

Settings is a small module which handles current user specific information. The main component contains a card with two tabs. The first tab is general preferences and here a nurse can update his or her names, avatar, email and phone number. Also, language settings are located in general preferences tab. The second tab serves the purpose of changing user password.

3.3 Server communication with HttpClient and RxJs

Most web applications need to communicate with a server over the HTTP protocol.

However, HttpClient is not a core Angular module but it is provided by the framework and can be used right away without searching for another third party http library. The module provides a simple set of APIs which enable making all HTTP requests. In addition, since HttpClient is a part of the Angular framework, it enables simple testing for http services, strong typing of request and response objects, better error handling with the help of observables as well as request and response interceptors. Service responses can be handled both with default browser APIs using Promise or using Observables from RxJS library. Moreover, the latter provides a set of powerful tools for data manipulation and aggregation, which is widely used throughout the whole project.

All modules described in section 3.2 have a set of services which retrieve the required data from a server and make use of it in module components. For example, the forms module has a set of services to retrieve questionnaires answered by patients. Listing 6 describes the interface of QuizService.

```
get(quizFilter: QuizFilter):
  Observable<PaginatedItemsResponse> {}
getOne(timelineEventId: string): Observable<Quiz> {}
getQuestions(quizId: string): Observable<QuizQuestion[]> {}
getCategoryScores(quizId: string):
  Observable<QuizCategoryScore[]> {}
getCategories(): Observable<QuizCategory[]> {}
```

Listing 6. QuizService interface

According to the listing 6, QuizService has five methods: get, getOne, getQuestions, getCategoryScores and getCategories. The first method allows the client to fetch all quizzes according to filtering criteria. The second function returns a single quiz fetched by its timeline event id. The third listing 6 entry retrieves all questions related to some particular questionnaire. The next method allows to obtain information about quiz scores for questions grouped by category. The last function fetches all available quiz categories. Moreover, most of the models in the application have their respective services for fetching data from a server.

After a service is implemented, it should be included in the list of providers for at most

one module of Angular application and later on a service can be injected into component. Framework utilizes dependency injection design pattern to reuse same service between different components. The main idea behind DI (dependency injection) is the fact that a class receives its dependencies from external sources instead of creating them by itself internally. In practice, Angular services are instantiated only once and then passed over dependency injection mechanism into different components.

3.4 Data storage and manipulation with Ngrx and Redux

Modern web applications have various ways to handle data storage. Some can avoid keeping any data inside browsers and request updates from server whenever it is required. However, highly interactive and data intensive applications cannot afford many expensive, in terms of data volumes, calls to a server. These types of applications need to store data locally in browser to deliver a better user experience. A few most common techniques used for front-end applications to store data were described in section 1.2. Consequently, Redux data storage pattern has been chosen for the subject application of this thesis after comparison analysis.

The general idea of Redux is relatively simple. It requires all data to be stored in a centralized manner in a single JavaScript object. However, each module of the application has its own dedicated object named Store. The latter is populated with default values when initialized and after that it provides access to all data needed for a particular module. An overview of the integration of Store with the app model is illustrated in figure 6. First, it is easy to notice that module components are not allowed to modify Store data directly. The only way to trigger a data change in Store is to dispatch an Action. It is represented by a class used to describe the change which a component needs to make to the Store. For example, if a component of OperationsModule wants to update an operation information in Store, then the signature of an Action class which needs to be dispatched could have a constructor shown in listing 7.

```
class UpdateOperationAction(
      operationId: string, operation: Operation)
```

Listing 7. Store Action class definition

Later, this action can be dispatched from any component which has Store instance injected as shown in listing 8.

```
this.store.dispatch(
      new UpdateOperationAction({
            operationId: 123,
            operation: op
      })
)
```

Listing 8. Dispatching Store Action

After Action has been dispatched the task of updating operation in Store is delegated to a Reducer. The latter is a pure function which does not directly modify value in store but it receives a Store object as an input together with a payload from Action and returns a copy of Store object with modified operation data.

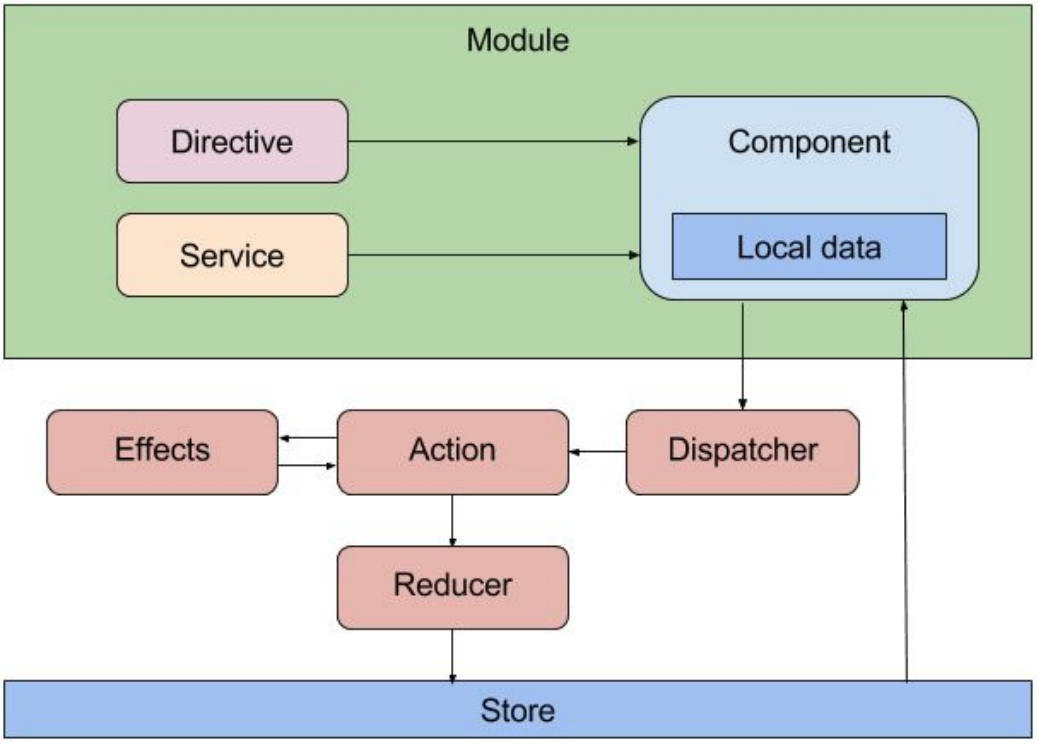

Figure 6. Store integration with application module

Another element of the figure 6 is Effects block. It is an interesting piece which complements the whole Store logic. In essence, an Action might not directly cause any changes to Store but it can cause one or many side effects which in turn are other Actions and they can cause Store state to be updated. These side effects are Effects, therefore the name. They play an important role in Store architecture and help to group and encapsulate related actions, therefore simplifying APIs for developer.

Next, in order to illustrate how Effects work, let us have a look at an example. The target application, a hospital dashboard, has a section where a hospital worker can review the submitted questionnaire answers. The questionnaire review page can be navigated from a list of submitted questionnaires or if user enters a url for some specific questionnaire directly into the browser search field. Consequently, in both cases user is going to end up on a questionnaire review page which contains some generic information about the quiz such as title, related operation name and patient names, date of submission and deadline date. Moreover, each quiz has questions and answers. In addition each questionnaire answer has a comments section where nurses can leave clarifications for some answers. In summary, in order to build a quiz review

page on the client-side application needs to have data for three entities: quiz, quiz answers and quiz answer comments. In turn, backend provides RESTful API endpoints for retrieving that data. The latter read only APIs are described in the following way:

- GET /quizzes/:id/ - retrieves short info about single questionnaire
- GET /quizzes/:id/questions/ - retrieves a list of questionnaire questions with answers
- GET /quiz-question/:id/comments/ - retrieves a list of comments for a single questionnaire answer

Then, the dashboard application needs to provide angular services to fetch corresponding data from the server. In this case it requires two services for questionnaire and for questionnaire question. Furthermore, class definitions for these services could be described as in listing 9.

```
@Injectable()
export class QuizService {

  /**
   * Retrieve single questionnaire by id
   * @param quizId {string}
   * @returns {Observable<Quiz>}
   */
  getOne(quizId: string): Observable<Quiz> {}

  /**
   * Retrieve questionnaire questions with answers
   * @param quizId {string}
   * @returns {QuizQuestion[]}
   */
  getQuestions(quizId: string): Observable<QuizQuestion[]> {}
}


@Injectable()
export class QuestionService {
```

```
  /**
   * Retrieve a list of comments for a single question
   * @param questId {string}
   * @returns {Observable<QuestionComment[]>}
   */
   getComments(questId: string): Observable<QuestionComment[]>
{}
}
```

Listing 9. Questionnaire and question services definitions

As we can see from listing 9, methods implementations are missing from the examples but they correspond exactly to the APIs provided by the backend. Going back to questionnaire review page, now we have to call three methods of two services to fetch data to display all needed information on the page. Thus this is the place where Effects module comes into play. Instead of remembering which methods of which services to call Effects allow developers to add a layer of abstraction and dispatch only a single action to the Store. For example, action could be FetchAllQuizInfo(quizId) where quizId is a current questionnaire id. Then that action will cause an Effect which in turn will make all necessary calls to our services, fetch all required data from server and dispatch another set of actions to set the data to Store.

Overall, the Redux data storage pattern and its implementation ngrx library for Angular projects proved to be reliable in terms of quality, helpful for developers in debugging the application, increased productivity and helped to structure data better for the client application.

4 Results

The main goal of this paper was to develop and describe a web application solution for Buddy Healthcare Oy. Thus, as a result of an almost one year project the hospital dashboard for monitoring patients was developed. The complete solution includes an Angular application which is hosted on AWS (Amazon Web Services). Also, the development process was established in such a way that there are two versions of the dashboard: one for development and quality assurance purposes and the second one for hospitals use. Moreover, continuous integration and delivery is automated and does not require much effort from a developer perspective except for pushing code to a remote git (version control system) repository hosted on GitHub.

As for hospitals, they are provided with access to the dashboard limited by patients belonging to that specific hospital only. Overall, the solution enables creating new surgeries with activation codes which are later distributed to up to four caretakers for a single patient. Hospital workers can monitor a patient's activity and interaction with the mobile application immediately after a new procedure has been created. However, the operation will stay in the Activation Code state until any user utilizes an activation code provided by the hospital. After the activation code has been used, nurses can receive information in the dashboard about patients. The information might include, for example, acknowledgments of reading some important information about the procedure, submitted pre-operative questionnaires, conversation messages, acknowledgments of postoperative exercise being done by patient.

Overall, the initial goal of the thesis was achieved, and the developed application is actively used in multiple hospitals around Finland.

5 Discussion

However, the goal of the project has been achieved successfully, more improvements and further developments are currently under discussion at Buddy Healthcare Oy. One of the biggest changes in the software is going to be a patient care path approach together with dynamic timelines for general physiotherapy treatment. The challenge in this case is to migrate from a more operation-centric solution to a more generic care path. The main difference is that the care path for patients might not include the operation itself and therefore can be applied to a wider range of procedures and treatment processes. For example, currently the whole operation procedure instructions and general information are contained within a single timeline of events which a mobile application user receives once he/she used activation code. On the contrary, the care path approach will allow hospital workers to construct multiple timelines in a single care path as well as add additional timelines later if needed. For example, we can define a care path for knee replacement surgery to contain one timeline for general information about the surgery itself, a second timeline for pre-operative appointment, a third timeline for preoperative and postoperative instructions and the last timeline for postoperative appointment. However, it might happen that a patient needs multiple postoperative appointments and maybe a series of medication which can be treated as separate timelines within the same care path and could be added on demand later on. Also, this kind of approach could be applied in general physiotherapy cases when a patient follows some timeline of instructions and at some point a doctor has to make a decision which timeline to follow next.

Overall, Buddy Healthcare Oy as a company is still in the early stages of developing a complete solution for an automating hospital-patient communication and treatment process. However, there are no obvious obstacles in resources or technologies available to help the company to achieve its ambitious goals and mission.

6 Conclusion

Overall, the main goal of the thesis to develop a dashboard web application for hospitals was achieved and a new solution for Buddy Healthcare Oy has been developed.

In conclusion, the application development process proceeded without major deviations. However, there was a certain risk when the Angular framework was selected for the project a year ago because it was not in the final release version but still in beta. However, the framework proved to be robust and reliable for enterprise application development and has matured over the year significantly. As for the schedule for this thesis project, it has deviated from the schedule for about a month due to the intensive development process of the dashboard itself.

Finally, the resulting product, a hospital web dashboard, is currently being actively used in multiple hospitals across Finland.

7 References

1. Deffree S., ARPANET establishes 1st computer-to-computer link, October 29, 1969, [online]. EDN Network, 29 October 2016
   URL:
   https://www.edn.com/electronics-blogs/edn-moments/4399541/ARPANET-establishes-1st-computer-to-computer-link--October-29--1969
   Accessed 6 October 2017
2. BBC, Finland makes broadband a 'legal right', [online]. BBC, 1 July 2010
   URL: http://www.bbc.com/news/10461048
   Accessed 6 October 2017
3. Carlström V., Finns consume crazy amounts of data - no other country even comes close, [online]. Business Insider Nordic, 21 December 2016
   URL: http://www.bbc.com/news/10461048
   Accessed 6 October 2017
4. Facebook Inc., React Documentation, [online], Facebook Inc., 8 October 2017
   URL: https://reactjs.org/docs/hello-world.html
   Accessed 8 October 2017
5. vuejs, Vue.js Documentation, [online], vuejs, 8 October 2017
   URL: https://vuejs.org/v2/guide/
   Accessed 8 October 2017
6. emberjs, Ember.js Documentation, [online], emberjs, 29 August 2015
   URL: https://guides.emberjs.com/v2.15.0/
   Accessed 8 October 2017
7. Google Inc., Angular Documentation, [online], Google Inc., 26 September 2017
   URL: https://angular.io/
   Accessed 6 October 2017
8. Savkin V., Angular: Why TypeScript?, [online], medium.com, 22 July 2016
   URL: https://vsavkin.com/writing-angular-2-in-typescript-1fa77c78d8e8
   Accessed 6 October 2017
9. Meteor, Meteor Documentation, [online], Meteor, 4 October 2017
   URL: https://guide.meteor.com/
   Accessed 8 October 2017
10. MDN Web docs, Storage, [online], developer.mozilla.org, 17 July 2017

URL: https://developer.mozilla.org/en-US/docs/Web/API/Storage
Accessed 6 October 2017

11. Facebook Inc., Flux, [online], Facebook Inc., 24 March 2017
URL: https://facebook.github.io/flux/docs/overview.html#content
Accessed 6 October 2017
12. React Community, Redux, [online], React Community, 3 October 2017
URL: http://redux.js.org/docs/introduction/ThreePrinciples.html
Accessed 6 October 2017
13. MDN Web docs, IndexedDB API, [online], developer.mozilla.org, 17 September 2017
URL: https://developer.mozilla.org/en/docs/Web/API/IndexedDB_API
Accessed 6 October 2017
14. pouchdb, PouchDB documentation, [online], pouchdb, 5 September 2017
URL: https://www.html5rocks.com/en/tutorials/offline/storage/
Accessed 6 October 2017
15. Google Inc., Lovefield is a relational database for web apps, [online], Google Inc., 25 February 2017
URL: https://google.github.io/lovefield/
Accessed 6 October 2017
16. wycats, Handlebars documentation, [online], wycats, 29 January 2017
URL: http://handlebarsjs.com/
Accessed 6 October 2017
17. Representational state transfer, [online], Wikipedia, 10 September 2017
URL: https://en.wikipedia.org/wiki/Representational_state_transfer
Accessed 6 October 2017
18. Fielding R. T., Representational State Transfer, [online], University of California, Irvine, 2000
URL: http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
Accessed 6 October 2017
19. Fette I., Google Inc., Melnikov A., Isode Ltd., The WebSocket Protocol, [online], Internet Engineering Task Force, Category: Standards Track, ISSN: 2070-1721, December 2011
URL: https://tools.ietf.org/html/rfc6455
Accessed 6 October 2017
20. MDN Web docs, WebSockets, [online], developer.mozilla.org, 17 September

2017
URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API
Accessed 6 October 2017

21. Facebook Inc., GraphQL Documentation at graphql.org, [online], Facebook Inc., 4 October 2017
URL: http://graphql.org/learn/
Accessed 6 October 2017
22. Facebook Inc., GraphQL, [online], Facebook Inc., October 2016
URL: https://facebook.github.io/graphql/
Accessed 6 October 2017